CERTORA

Formal Verification of Compound Protocol

Certora Team

Summary

This report presents work and conclusions based on a collaboration between Compound and Certora using Certora's Prover tool to formally verify security properties of Compound's Money Market smart contracts. We provided Compound with Certora Prover, which is a tool for formally verifying that smart contracts satisfy specifications written in a language called Specify.

The Compound team worked with Certora to build specifications of the protocol. The main focus of the specifications was to ensure that funds are correctly tracked as they move in and out of the Compound protocol (i.e. funds are neither created from thin air nor destroyed). The Certora Prover tool verified that Compound's implementation of the protocol satisfies these specifications.

Certora Prover was integrated inside Compound's continuous integration system to verify these properties throughout the contracts' development cycle. The latest commit that was monitored is f385d71983ae5c5799faae9b2dfea43e5cf75262. The initial project was held from April 9th, 2019 until May 9th 2019, but the specifications developed during this period continue to be used to verify later versions of the code.

The Certora Prover proved the Compound Protocol implementation correct with respect to its specification. During the verification process, Certora Prover discovered a number of issues, which were promptly corrected prior to releasing the protocol.

Disclaimer

Certora Prover takes as input a contract and a specification and formally proves that the contract satisfies the specification in all scenarios. Importantly, the guarantees of Certora Prover are scoped to the provided specification, and any cases not covered by the specification are not checked by Certora Prover.

We hope that this information is useful, but provide no warranty of any kind, express or implied. The contents of this report should not be construed as a complete guarantee that the Compound system is secure in all dimensions. In no event shall Certora or any of its employees be liable for any claim,

damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the results reported here.

Methodology

The Certora Prover tool checks contracts in a modular fashion— every deployed address is checked on its own. For example, cToken was checked separately from the Comptroller and the concrete ERC20 implementation. To correctly model cToken's dependencies on the Comptroller, we wrote a verification harness that models any possible value that can be returned from a Comptroller method. For the ERC20, we took both (1) a reference implementation as a harness and (2) ignored the actual implementation to uncover ERC20 behaviors that might break the Money Market assumptions.

Properties Verified

- Implementations of the following cToken functions match the Compound Specification for all possible inputs.
  - Accrue Interest - A function to calculate interest on borrows
  - Mint - A function to provide underlying and receive cTokens
  - Redeem - A function to provide cTokens and receive underlying
  - Borrow - A function to borrow from the protocol
  - Repay Borrow - A function to repay a borrow to the protocol
  - Liquidate - A function for liquidation of underwater accounts
  - Seize - Internal function used by liquidation
- cToken's `transferFrom` method correctly updates balances in accordance with the ERC20 standard for all possible inputs.
- Each cToken function modifies only the fields it is expected to modify.
- Unitroller's administrative functions for setting and accepting an administrator, setting and accepting an implementation match the Compound Specification.
- The Maximillion utility contract for repaying cEther borrows is functioning correctly for all inputs.

Issues Discovered and Patched

The process of running the tool and analyzing the resulting bug reports uncovered the following concerns:

- The cToken contract must not have the role of minter, redeemer, borrower or liquidator.
- When exchanging tokens to cTokens, a nonzero number of tokens should never produce zero cTokens.
- More generally, rounding error in the exchange rate conversion should be bounded.

- ERC20 tokens added as markets should adhere to certain rules to be fully compatible with the Compound code. One such rule is to avoid changing any state in standard ERC20 methods when they return false.
    - These properties can be checked on the underlying ERC20 contract by the Certora Prover, as well.
- A previous version of Maximillion incorrectly used stored borrows instead of current borrows in the computation of interest accrual. The fixed version of Maximillion was verified to ensure it was always possible to repay borrows.

In all cases, we found that the Compound team promptly addressed these issues.

Technology overview

Certora Prover is based on well-studied techniques from the formal verification community. Specifications define a set of rules that call into the contract under analysis and make various assertions about their behavior. These rules, together with the contract under analysis, are compiled to a logical formula called a verification condition, which is then proved or disproved by the solver Z3. If the rule is disproved, the solver also provides a concrete test case demonstrating the violation.

The rules of the specification play a crucial role in the analysis. Without good rules, only very shallow properties can be checked (e.g. that no assertions in the contract itself are violated). To make effective use of Certora Prover, users must write rules that describe the high-level properties they wish to check of their contract. Certora Prover cannot make any guarantees about cases that fall outside the scope of the rules provided to it as input. Thus, in order to understand the results of this analysis, one must carefully understand the specification's rules.

Conclusion

The verification of the Compound Protocol succeeded in proving that the implementation correctly matches the specification. During the process, several flaws in earlier versions of the implementation were discovered and fixed. Overall, Certora Prover's verification increased confidence in the security of the Compound Protocol.

Appendix

This appendix includes a partial list of properties checked by the Certora Prover for the Compound Protocol.

Accrue Interest

accrueInterest.cvl

```
methods {
    getCash() returns uint256
    accrueInterest() returns uint256
    interestRateModelGetBorrowRate() returns uint256, uint256
    accrualBlockNumber() returns uint256
    borrowIndex() returns uint256
    totalBorrows() returns uint256
    totalReserves() returns uint256
    reserveFactorMantissa() returns uint256
}

accrueInterest(uint result)
description "Break accrueInterest with result=$result, block delta is $delta" {
    // Pre/action/post environments
    env e0; havoc e0; // pre+action
    env e1; havoc e1; // post

    static_require e1.block.number >= e0.block.number;

    // fetch pre
    uint256 cTokenCashPre = sinvoke getCash(e0);
    uint256 interestRateModelErr;
    uint256 borrowRateMantissaPre;
    interestRateModelErr, borrowRateMantissaPre = sinvoke interestRateModelGetBorrowRate(e0);
    uint256 accrualBlockNumberPre = sinvoke accrualBlockNumber(e0);
    uint256 borrowIndexPre = sinvoke borrowIndex(e0);
    uint256 totalBorrowsPre = sinvoke totalBorrows(e0);
    uint256 totalReservesPre = sinvoke totalReserves(e0);
    uint256 reserveFactorPre = sinvoke reserveFactorMantissa(e0);

    // internal computations
    uint256 delta = e0.block.number-accrualBlockNumberPre;
    uint256 simpleInterestFactor = delta*borrowRateMantissaPre;
    uint256 interestAccumulated = (totalBorrowsPre * simpleInterestFactor) / 1000000000000000000;

    // post expected
    uint256 borrowIndexPostExpected = borrowIndexPre +
```

```
(borrowIndexPre*simpleInterestFactor)/1000000000000000000;
    uint256 totalBorrowsPostExpected = totalBorrowsPre + interestAccumulated;
    uint256 totalReservesPostExpected = totalReservesPre +
(interestAccumulated*reserveFactorPre)/1000000000000000000;

    // Action!
    static_require result == invoke accrueInterest(e0);
    bool accrueInterestReverted = lastReverted;

    // fetch post
    uint256 accrualBlockNumberPost = sinvoke accrualBlockNumber(e1);
    uint256 borrowIndexPostActual = sinvoke borrowIndex(e1);
    uint256 totalBorrowsPostActual = sinvoke totalBorrows(e1);
    uint256 totalReservesPostActual = sinvoke totalReserves(e1);

    uint256 NO_ERROR = 0;
    uint256 INTEREST_RATE_MODEL_ERROR = 5;
    uint256 MATH_ERROR = 9;

    // Guarantee return values
    static_assert (!accrueInterestReverted) =>
        (result == NO_ERROR ||
         result == INTEREST_RATE_MODEL_ERROR ||
         result == MATH_ERROR), "Got unexpected error code $result";

    static_assert (!accrueInterestReverted => (result == INTEREST_RATE_MODEL_ERROR <=>
interestRateModelErr != 0)), "Mismatch in case of interest rate model error";

    static_assert (!accrueInterestReverted =>
                   ((result != 0 || delta == 0) <=>
                    (accrualBlockNumberPost == accrualBlockNumberPre &&
                     borrowIndexPostActual == borrowIndexPre &&
                     totalBorrowsPostActual == totalBorrowsPre &&
                     totalReservesPostActual == totalReservesPre))), "Mismatch in error case";

    static_assert (!accrueInterestReverted =>
                   ((result == 0) <=>
                    (accrualBlockNumberPost == e0.block.number &&
                     borrowIndexPostActual == borrowIndexPostExpected &&
                     totalBorrowsPostActual == totalBorrowsPostExpected &&
                     totalReservesPostActual == totalReservesPostExpected
                     ))), "Mismatch in no error case: borrowIndex: pre=$borrowIndexPre,
actual=$borrowIndexPostActual, expected=$borrowIndexPostExpected; totalBorrows:
pre=$totalBorrowsPre, actual=$totalBorrowsPostActual, expected=$totalBorrowsPostExpected;
totalReserves: pre=$totalReservesPre, acutal=$totalReservesPostActual,
expected=$totalReservesPostExpected";
```

```
}
```

Mint and Redeem

mintAndRedeemFresh.cvl

```
methods {
        admin() returns address
        totalSupply() returns uint256
        balanceOf(address) returns uint256
        accrualBlockNumber() returns uint256
        initialExchangeRateMantissa() returns uint256
        totalBorrows() returns uint256
        totalReserves() returns uint256

        // Exposing internal functions
         mintFreshPub(address,uint256) returns uint256
         redeemFreshPub(address,uint256,uint256) returns uint256

         exchangeRateStoredInternalPub() returns uint256, uint256
         checkTransferInPub(address,uint256) returns uint256
         doTransferInPub(address,uint256) returns uint256
         getCash() returns uint256
        comptrollerMintAllowed(address,address,uint256) returns uint256
        comptrollerRedeemAllowed(address,address,uint256) returns uint256

        // Simulation of functions that have effects on external contracts
        doTransferInPubSim(address,uint256) returns uint256
        doTransferOutPubSim(address,uint256) returns uint256

        // Lemmas
        cTokenMintComputation(uint256) returns uint256
        cTokenRedeemComputation(uint256) returns uint256
}

rule mintFresh(uint result, address account, uint256 amount)
description "Failed to mint fresh asset with result $result (minter account=$account,
amount=$amount)" {

        // Free Variables
        env e0; havoc e0; // pre
        env e1; havoc e1; // invocation
        env e1b; havoc e1b; // invocations on behalf of CToken
        env e2; havoc e2; // post

        // Strict ordering
        static_require e1.block.number >= e0.block.number;
```

```
        static_require e2.block.number >= e1.block.number;

        // Capture current values
        uint256 totalSupplyPre = sinvoke totalSupply(e0);
        uint256 accountSupplyPre = sinvoke balanceOf(e0, account);
        uint256 marketBlockNumber = sinvoke accrualBlockNumber(e0);

        // Simulate checks that depend on external contracts
        uint comptrollerCheckResult = sinvoke comptrollerMintAllowed(e1,
currentContract,account,amount);
        uint labelCheckTransferIn = sinvoke checkTransferInPub(e1b, account, amount);
        uint labelDoTransferIn = sinvoke doTransferInPubSim(e1, account, amount); // SG: make sure
doTransferInPubSim and doTransferIn return the same value when starting from the same state, and
make sure dummy can simulate all observable external behaviors of the call.

        // Invoke mintFresh
        static_require result == invoke mintFreshPub(e1, account, amount);
        bool mintFreshReverted = lastReverted;

        // Get next values
        uint256 totalSupplyPost = sinvoke totalSupply(e2);
        uint256 accountSupplyPost = sinvoke balanceOf(e2, account);

        // Helper for checking balance has not changed
        bool staticBalance =
               ( totalSupplyPost == totalSupplyPre &&
                 accountSupplyPost == accountSupplyPre );

        // Helper for checking balance has not changed, as expected.
        // Precise change is computed in cTokenComputationLemma
        bool dynamicBalance = (totalSupplyPost - totalSupplyPre == accountSupplyPost -
accountSupplyPre) // Change in totalSupply is same as change in accountSupply
                                          && (totalSupplyPost - totalSupplyPre >= 0); // The
change in supplies must be >= 0

        bool comptrollerCheckSuccess = comptrollerCheckResult == 0;
        bool accrued = marketBlockNumber == e1.block.number;
        bool checkTransferSuccess = labelCheckTransferIn == 0;
        bool doTransferSuccess = labelDoTransferIn == 0;

        // Track error codes
        uint256 NO_ERROR = 0;
        uint256 COMPTROLLER_REJECTION = 3;
        uint256 NOT_ACCRUED = 10;
        uint256 CHECK_TRANSFER_FAILED_1 = 12;
        uint256 CHECK_TRANSFER_FAILED_2 = 13;
```

```
        uint256 TOKEN_TRANSFER_IN_FAILED = 15;
        uint256 MATH_ERROR = 9;

        // Guarantee return values
        static_assert (!mintFreshReverted && result != MATH_ERROR) => (
          result == NO_ERROR ||
          result == COMPTROLLER_REJECTION ||
          result == NOT_ACCRUED ||
          result == CHECK_TRANSFER_FAILED_1 ||
          result == CHECK_TRANSFER_FAILED_2 ||
          result == TOKEN_TRANSFER_IN_FAILED ||
          result == MATH_ERROR
        ), "Got unexpected error code $result";

        /* All these cases depend on: (1) mintFresh not reverting; and (2) the error code being
anything  other than a math error.
               The idea is that (1) reverts, if are not catched by the caller (could it be a
contract by Compound?), will have no effect on the state;
               and (2) math errors, which are handled separately.
         */
        // Success case updates market accordingly
        static_assert (!mintFreshReverted && result != MATH_ERROR) => (result == NO_ERROR <=> (
               comptrollerCheckSuccess &&
               accrued &&
               checkTransferSuccess &&
               doTransferSuccess &&
               dynamicBalance
        )), "Mismatch in no error case (0). Got result $result";

        // Policy hook rejected case
        static_assert (!mintFreshReverted && result != MATH_ERROR) => (result ==
COMPTROLLER_REJECTION <=> (
               !comptrollerCheckSuccess &&
               staticBalance
        )), "Mismatch in comptroller rejection case (3). Got result $result";

        // Not accrued
        static_assert (!mintFreshReverted && result != MATH_ERROR) => (result == NOT_ACCRUED <=> (
               comptrollerCheckSuccess &&
               !accrued &&
               staticBalance
        )), "Mismatch in non-accrued case (10=0xa). Got result $result";

        // Check transfer failure
        static_assert (!mintFreshReverted && result != MATH_ERROR) => ((result ==
CHECK_TRANSFER_FAILED_1 || result == CHECK_TRANSFER_FAILED_2) <=> (
```

```
                comptrollerCheckSuccess &&
                accrued &&
                !checkTransferSuccess &&
                staticBalance
        )), "Mismatch in check transfer failure (12,13). Got result $result";

        // Do transfer failure
        static_assert (!mintFreshReverted && result != MATH_ERROR) => (result ==
TOKEN_TRANSFER_IN_FAILED <=>
        (
                comptrollerCheckSuccess &&
                accrued &&
                checkTransferSuccess &&
                !doTransferSuccess &&
                staticBalance
        )), "Mismatch in do transfer failure (15). Got result $result";

        // Math error -> no state changes (at least, the ones we observe here! This is not checking
about the FULL state of the contract)
        static_assert !mintFreshReverted => (result == MATH_ERROR =>
        (
                staticBalance
        )), "State changes occuring despite math error, totalSupplyPre=$totalSupplyPre,
totalSupplyPost=$totalSupplyPost ; accountSupplyPre=$accountSupplyPre,
accountSupplyPost=$accountSupplyPost";
}

rule mintFresh_extended(uint result, address account, uint256 amount)
description "Failed to mint fresh asset with result $result (account=$account, amount=$amount)" {
        // Free Variables
        env e0; havoc e0; // pre
        env e1; havoc e1; // invocation
        env e2; havoc e2; // post

        // Strict ordering
        static_require e1.block.number >= e0.block.number;
        static_require e2.block.number >= e1.block.number;

         // Minting to self is assumed to be impossible
         static_require account != currentContract;

        // Capture current values
        uint256 totalSupplyPre = sinvoke totalSupply(e0);
        uint256 accountSupplyPre = sinvoke balanceOf(e0, account);
        uint256 marketBlockNumber = sinvoke accrualBlockNumber(e0);
```

```
        uint256 cash = sinvoke getCash(e0);

        // Label CALL results
        uint comptrollerCheckResult = sinvoke comptrollerMintAllowed(e1, currentContract, account,
amount);
        uint labelCheckTransferIn = sinvoke checkTransferInPub(e1, account, amount);
        uint labelDoTransferIn = sinvoke doTransferInPubSim(e1, account, amount);

        // Invoke mintFresh
        static_require result == invoke mintFreshPub(e1, account, amount);
        bool mintFreshReverted = lastReverted;

        // Get next values
        uint256 totalSupplyPost = sinvoke totalSupply(e2);
        uint256 accountSupplyPost = sinvoke balanceOf(e2, account);
        uint256 newCash = sinvoke getCash(e2);

        // Helper for checking balance has not changed
        bool staticBalance =
                ( totalSupplyPost == totalSupplyPre &&
                  accountSupplyPost == accountSupplyPre )
                  && cash == newCash ;

        // Helper for checking balance has not changed, as expected.
        // Precise change is computed in cTokenComputationLemma
        bool dynamicBalance = (totalSupplyPost - totalSupplyPre == accountSupplyPost -
accountSupplyPre) // Change in totalSupply is same as change in accountSupply
                && (totalSupplyPost - totalSupplyPre >= 0) // The change in supplies must be >= 0
                && (newCash - cash == amount); // The change in cash must be amount

        bool comptrollerCheckSuccess = comptrollerCheckResult == 0; // Comptroller is an external
contract and its error is converted to COMPTROLLER_REJECTION, so no need to require on the label
        bool accrued = marketBlockNumber == e1.block.number;
        bool checkTransferSuccess = labelCheckTransferIn == 0;
        bool doTransferSuccess = labelDoTransferIn == 0;

        // Track error codes
        uint256 NO_ERROR = 0;
        uint256 COMPTROLLER_REJECTION = 3;
        uint256 NOT_ACCRUED = 10;
        uint256 CHECK_TRANSFER_FAILED_1 = 12;
        uint256 CHECK_TRANSFER_FAILED_2 = 13;
        uint256 TOKEN_TRANSFER_IN_FAILED = 15;
        uint256 MATH_ERROR = 9;

        // Guarantee return values
```

```
        static_assert (!mintFreshReverted && result != MATH_ERROR) => (
          result == NO_ERROR ||
          result == COMPTROLLER_REJECTION ||
          result == NOT_ACCRUED ||
          result == CHECK_TRANSFER_FAILED_1 ||
          result == CHECK_TRANSFER_FAILED_2 ||
          result == TOKEN_TRANSFER_IN_FAILED ||
          result == MATH_ERROR
        ), "Got unexpected error code $result";

        /* All these cases depend on: (1) mintFresh not reverting; and (2) the error code being
anything  other than a math error.
              The idea is that if (1) reverts, unless catched by the caller (could it be a contract
by Compound?), will have no effect on the state;
              and (2) math errors, which are handled separately without reverting, do not lead to
state changes.
         */
        // Success case updates market accordingly
        static_assert (!mintFreshReverted && result != MATH_ERROR) => (result == NO_ERROR <=> (
               comptrollerCheckSuccess &&
               accrued &&
               checkTransferSuccess &&
               doTransferSuccess &&
               dynamicBalance
        )), "Mismatch in no error case (0). Got result $result";

        // Policy hook rejected case
        static_assert (!mintFreshReverted && result != MATH_ERROR) => (result ==
COMPTROLLER_REJECTION <=> (
               !comptrollerCheckSuccess &&
               staticBalance
        )), "Mismatch in comptroller rejection case (3). Got result $result";

        // Not accrued
        static_assert (!mintFreshReverted && result != MATH_ERROR) => (result == NOT_ACCRUED <=> (
               comptrollerCheckSuccess &&
               !accrued &&
               staticBalance
        )), "Mismatch in non-accrued case (10=0xa). Got result $result";

        // Check transfer failure
        static_assert (!mintFreshReverted && result != MATH_ERROR) => ((result ==
CHECK_TRANSFER_FAILED_1 || result == CHECK_TRANSFER_FAILED_2) <=> (
               comptrollerCheckSuccess &&
               accrued &&
               !checkTransferSuccess &&
```

```
                staticBalance
        )), "Mismatch in check transfer failure (12,13). Got result $result";

        // Do transfer failure
        static_assert (!mintFreshReverted && result != MATH_ERROR) => (result ==
TOKEN_TRANSFER_IN_FAILED <=>
        (
                comptrollerCheckSuccess &&
                accrued &&
                checkTransferSuccess &&
                !doTransferSuccess &&
                staticBalance
        )), "Mismatch in do transfer failure (15). Got result $result";

        // Math error -> no state changes (at least, the ones we observe here! This is not checking
about the FULL state of the contract)
        static_assert !mintFreshReverted => (result == MATH_ERROR =>
        (
                staticBalance
        )), "State changes occuring despite math error, totalSupplyPre=$totalSupplyPre,
totalSupplyPost=$totalSupplyPost ; accountSupplyPre=$accountSupplyPre,
accountSupplyPost=$accountSupplyPost";
}


rule redeemFresh(uint result, address account, uint256 amountCTokens, uint256 amountUnderlying)
description "Failed to redeemFresh fresh asset with result $result (account=$account,
amountCTokens=$amountCTokens, amountUnderlying=$amountUnderlying)" {
        // Free Variables
        env e0; havoc e0; // pre
        env e1; havoc e1; // invocation
        env e1b; havoc e1b; // invocations on behalf of CToken
        env e2; havoc e2; // post

        // Strict ordering
        static_require e1.block.number >= e0.block.number;
        static_require e2.block.number >= e1.block.number;

         // Redeeming from self is assumed to be impossible
         static_require account != currentContract;

        // Capture current values
        uint256 totalSupplyPre = sinvoke totalSupply(e0);
        uint256 accountSupplyPre = sinvoke balanceOf(e0, account);
        uint256 marketBlockNumber = sinvoke accrualBlockNumber(e0);
        uint256 cash = sinvoke getCash(e0);
```

```
        // Simulate checks that depend on external contracts
        uint comptrollerCheckResult = sinvoke comptrollerRedeemAllowed(e1, currentContract, account,
amountCTokens); // Should include amountUnderlying?
        uint labelDoTransferOut = sinvoke doTransferOutPubSim(e1, account, amountUnderlying);

        // Invoke mintFresh
        static_require result == invoke redeemFreshPub(e1, account, amountCTokens, amountUnderlying);
        bool redeemFreshReverted = lastReverted;

        // Get next values
        uint256 totalSupplyPost = sinvoke totalSupply(e2);
        uint256 accountSupplyPost = sinvoke balanceOf(e2, account);
        uint256 newCash = sinvoke getCash(e2);

        // Helper for checking balance has not changed
        bool staticBalance =
                ( totalSupplyPost == totalSupplyPre &&
                  accountSupplyPost == accountSupplyPre )
                  && cash == newCash ;

        // Helper for checking balance has not changed, as expected.
        // Precise change is computed in cTokenComputationLemma
        bool dynamicBalanceBasic = (totalSupplyPre - totalSupplyPost == accountSupplyPre -
accountSupplyPost) // Change in totalSupply is same as change in accountSupply
                && (totalSupplyPre - totalSupplyPost >= 0) // The change in supplies must be <= 0
                && (cash - newCash >= 0); // This operation transfers out
        bool dynamicBalanceRedeemingUnderlying = (cash - newCash == amountUnderlying); // The change
in cash must be amount
        bool dynamicBalanceRedeemingCToken = (totalSupplyPre-totalSupplyPost == amountCTokens); //
The change in CToken must be -amountCTokens
        bool dynamicBalanceAdvanced = ((amountCTokens > 0 && amountUnderlying == 0) =>
dynamicBalanceRedeemingCToken)
                && ((amountCTokens == 0 && amountUnderlying > 0) =>
dynamicBalanceRedeemingUnderlying)
                && ((amountCTokens == 0 && amountUnderlying == 0) => (dynamicBalanceRedeemingCToken
&& dynamicBalanceRedeemingUnderlying))
                && !(amountCTokens > 0 && amountUnderlying > 0);

        bool comptrollerCheckSuccess = comptrollerCheckResult == 0;
        bool accrued = marketBlockNumber == e1.block.number;
        bool checkTransferSuccess = cash >= amountUnderlying;
        bool doTransferSuccess = labelDoTransferOut == 0;

        // Track error codes
        uint256 NO_ERROR = 0;
```

```
        uint256 COMPTROLLER_REJECTION = 3;
        uint256 NOT_ACCRUED = 10;
        uint256 CHECK_TRANSFER_FAILED = 14; // TOKEN_INSUFFICIENT_CASH
        uint256 TOKEN_TRANSFER_OUT_FAILED = 16;
        uint256 MATH_ERROR = 9;

        // Guarantee return values
        static_assert (!redeemFreshReverted && result != MATH_ERROR) => (
          result == NO_ERROR ||
          result == COMPTROLLER_REJECTION ||
          result == NOT_ACCRUED ||
          result == CHECK_TRANSFER_FAILED ||
          result == TOKEN_TRANSFER_OUT_FAILED ||
          result == MATH_ERROR
        ), "Got unexpected error code $result";


        // Success case updates market accordingly
        static_assert (!redeemFreshReverted && result != MATH_ERROR) => (result == NO_ERROR <=> (
                comptrollerCheckSuccess &&
                accrued &&
                checkTransferSuccess &&
                doTransferSuccess &&
                dynamicBalanceBasic &&
                dynamicBalanceAdvanced
        )), "Mismatch in no error case (0). Got result $result";

        // Policy hook rejected case
        static_assert (!redeemFreshReverted && result != MATH_ERROR) => (result ==
COMPTROLLER_REJECTION <=> (
                !comptrollerCheckSuccess &&
                staticBalance
        )), "Mismatch in comptroller rejection case (3=0x3). Got result $result";

        // Not accrued
        static_assert (!redeemFreshReverted && result != MATH_ERROR) => (result == NOT_ACCRUED <=> (
                comptrollerCheckSuccess &&
                !accrued &&
                staticBalance
        )), "Mismatch in non-accrued case (10=0xa). Got result $result";

        // Check transfer failure
        // Note that since if amountUnderlying == 0 we do not compute in the spec the computed
amountUnderlying, so we skip the test for now
        static_assert (!redeemFreshReverted && result != MATH_ERROR && amountUnderlying > 0) =>
((result == CHECK_TRANSFER_FAILED) <=> (
```

```
                comptrollerCheckSuccess &&
                accrued &&
                !checkTransferSuccess &&
                staticBalance
        )), "Mismatch in check transfer failure (14). Got result $result";

        // Do transfer failure
        static_assert (!redeemFreshReverted && result != MATH_ERROR) => (result ==
TOKEN_TRANSFER_OUT_FAILED <=>
        (
                comptrollerCheckSuccess &&
                accrued &&
                checkTransferSuccess &&
                !doTransferSuccess &&
                staticBalance
        )), "Mismatch in do transfer failure (16). Got result $result";

        // Math error -> no state changes (at least, the ones we observe here! This is not checking
about the FULL state of the contract)
        static_assert !redeemFreshReverted => (result == MATH_ERROR =>
        (
                staticBalance &&
                newCash == cash
        )), "State changes occuring despite math error, totalSupplyPre=$totalSupplyPre,
totalSupplyPost=$totalSupplyPost ; accountSupplyPre=$accountSupplyPre,
accountSupplyPost=$accountSupplyPost";
}

rule mintThenRedeem(address account, uint256 amountUnderlying)
description "Mint and redeem are not inverses for account $account, amount $amountUnderlying"
{
        env e0; havoc e0;

        uint origCash = sinvoke getCash(e0);

        // both calls are "fresh"
        uint resultMint = invoke mintFreshPub(e0, account, amountUnderlying);
        bool revertedMint = lastReverted;

        uint resultRedeem = invoke redeemFreshPub(e0, account, 0, amountUnderlying);
        bool revertedRedeem = lastReverted;

        uint newCash = sinvoke getCash(e0);

        static_assert (resultMint == 0 && !revertedMint && resultRedeem == 0 && !revertedRedeem) =>
newCash == origCash;
```

```
}
```

Borrow and Repay

borrowAndRepayFresh.cvl

```
methods {
    totalSupply() returns uint256
    totalBorrows() returns uint256
    totalReserves() returns uint256

    balanceOf(address) returns uint256
    borrowBalanceStored(address) returns uint256
    exchangeRateStored() returns uint256

    getCash() returns uint256
    getCashOf(address) returns uint256 // not part of API

    borrowFreshPub(address, uint256) returns uint256
    repayBorrowFreshPub(address, address, uint256) returns uint256

    checkTransferInPub(address,uint256) returns uint256
    doTransferInPubSim(address,uint256) returns uint256
}

rule borrowFresh(uint result, address borrower, uint256 borrowAmount)
description "Break borrow with result=$result borrower=$borrower borrowAmount=$borrowAmount" {
    // Pre/action/post environments
    env e0; havoc e0;
    env e1; havoc e1;
    env e2; havoc e2;

    static_require e1.block.number >= e0.block.number;
    static_require e2.block.number >= e1.block.number;

    // Any other account
    address other; havoc other;
    static_require other != borrower && other != currentContract;

    /*
      - exchange rate should not change
      - errors should have no effect
      - no *other* storage should change - XXX can we specify this?

      |----------+--------+----------+-------|
      |          | CToken | Borrower | Other |
      |----------+--------+----------+-------|
```

```
  | cash     |   -A   |    +A    |   0   |
  | borrows  |   +A   |    +A    |   0   |
  | tokens   |    0   |     0    |   0   |
  | reserves |    0   |          |       |
  |----------+--------+----------+-------|
*/

/* Pre */

uint256 exchangeRatePre = sinvoke exchangeRateStored(e0);

uint256 cTokenCashPre = sinvoke getCash(e0);
uint256 borrowerCashPre = sinvoke getCashOf(e0, borrower);
uint256 otherCashPre = sinvoke getCashOf(e0, other);

uint256 cTokenBorrowsPre = sinvoke totalBorrows(e0);
uint256 borrowerBorrowsPre = sinvoke borrowBalanceStored(e0, borrower);
uint256 otherBorrowsPre = sinvoke borrowBalanceStored(e0, other);

uint256 cTokenTokensPre = sinvoke totalSupply(e0);
uint256 borrowerTokensPre = sinvoke balanceOf(e0, borrower);
uint256 otherTokensPre = sinvoke balanceOf(e0, other);

uint256 cTokenReservesPre = sinvoke totalReserves(e0);

// Just Do It
static_require result == invoke borrowFreshPub(e1, borrower, borrowAmount);
bool borrowFreshReverted = lastReverted;

/* Post */

uint256 exchangeRatePost = sinvoke exchangeRateStored(e2);

uint256 cTokenCashPost = sinvoke getCash(e2);
uint256 borrowerCashPost = sinvoke getCashOf(e2, borrower);
uint256 otherCashPost = sinvoke getCashOf(e2, other);

uint256 cTokenBorrowsPost = sinvoke totalBorrows(e2);
uint256 borrowerBorrowsPost = sinvoke borrowBalanceStored(e2, borrower);
uint256 otherBorrowsPost = sinvoke borrowBalanceStored(e2, other);

uint256 cTokenTokensPost = sinvoke totalSupply(e2);
uint256 borrowerTokensPost = sinvoke balanceOf(e2, borrower);
uint256 otherTokensPost = sinvoke balanceOf(e2, other);

uint256 cTokenReservesPost = sinvoke totalReserves(e2);
```

```
    // Measure
    bool staticBalance =
        (exchangeRatePost == exchangeRatePre) &&
        (cTokenCashPost == cTokenCashPre) &&
        (cTokenBorrowsPost == cTokenBorrowsPre) &&
        (cTokenTokensPost == cTokenTokensPre) &&
        (cTokenReservesPost == cTokenReservesPre) &&
        (borrowerCashPost == borrowerCashPre) &&
        (borrowerBorrowsPost == borrowerBorrowsPre) &&
        (borrowerTokensPost == borrowerTokensPre) &&
        (otherCashPost == otherCashPre) &&
        (otherBorrowsPost == otherBorrowsPre) &&
        (otherTokensPost == otherTokensPre);

    bool dynamicBalance =
        (borrowAmount != 0) &&
        (exchangeRatePost == exchangeRatePre) &&
        (cTokenCashPost == cTokenCashPre - borrowAmount) &&
        (cTokenBorrowsPost == cTokenBorrowsPre + borrowAmount) &&
        (cTokenTokensPost == cTokenTokensPre) &&
        (cTokenReservesPost == cTokenReservesPre) &&
        (borrowerCashPost == borrowerCashPre + borrowAmount) &&
        (borrowerBorrowsPost == borrowerBorrowsPre + borrowAmount) &&
        (borrowerTokensPost == borrowerTokensPre) &&
        (otherCashPost == otherCashPre) &&
        (otherBorrowsPost == otherBorrowsPre) &&
        (otherTokensPost == otherTokensPre);

    // XXX would be nice to pull named enums from actual contract?
    static_assert (!borrowFreshReverted =>
                   ((result != 0 || borrowAmount == 0) <=> staticBalance)), "Mismatch in static
case";
    static_assert (!borrowFreshReverted =>
                   ((result == 0 && borrowAmount != 0 && borrower != currentContract) <=>
dynamicBalance)), "Mismatch in dynamic case";
}

rule repayBorrowFresh(uint result, address payer, address borrower, uint256 repayAmount)
description "Break repay borrow with realRepayAmount=$realRepayAmount
borrowerBorrowsPre=$borrowerBorrowsPre" {
    // Pre/action/post environments
    env e0; havoc e0;
    env e1; havoc e1;
    env e2; havoc e2;
```

```
static_require e1.block.number >= e0.block.number;
static_require e2.block.number >= e1.block.number;

// Any other account
address other; havoc other;
static_require other != payer && other != borrower && other != currentContract;

// We assume this cannot happen
//  OQ: should we enforce in the code such that it need not be assumed?
static_require borrower != currentContract;

/*
  |----------+--------+-------+----------+-------|
  |          | CToken | Payer | Borrower | Other |
  |----------+--------+-------+----------+-------|
  | cash     |   +A   |  -A   |   -A/0   |   0   |
  | borrows  |   -A   | -A/0  |    -A    |   0   |
  | tokens   |    0   |   0   |     0    |   0   |
  | reserves |    0   |       |          |       |
  |----------+--------+-------+----------+-------|
*/

/* Pre */

uint256 exchangeRatePre = sinvoke exchangeRateStored(e0);

uint256 cTokenCashPre = sinvoke getCash(e0);
uint256 payerCashPre = sinvoke getCashOf(e0, payer);
uint256 borrowerCashPre = sinvoke getCashOf(e0, borrower);
uint256 otherCashPre = sinvoke getCashOf(e0, other);

uint256 cTokenBorrowsPre = sinvoke totalBorrows(e0);
uint256 payerBorrowsPre = sinvoke borrowBalanceStored(e0, payer);
uint256 borrowerBorrowsPre = sinvoke borrowBalanceStored(e0, borrower);
uint256 otherBorrowsPre = sinvoke borrowBalanceStored(e0, other);

uint256 cTokenTokensPre = sinvoke totalSupply(e0);
uint256 payerTokensPre = sinvoke balanceOf(e0, payer);
uint256 borrowerTokensPre = sinvoke balanceOf(e0, borrower);
uint256 otherTokensPre = sinvoke balanceOf(e0, other);

uint256 cTokenReservesPre = sinvoke totalReserves(e0);

// Just Do It
static_require result == invoke repayBorrowFreshPub(e1, payer, borrower, repayAmount);
bool repayBorrowFreshReverted = lastReverted;
```

```
    /* Post */

    uint256 exchangeRatePost = sinvoke exchangeRateStored(e2);

    uint256 cTokenCashPost = sinvoke getCash(e2);
    uint256 payerCashPost = sinvoke getCashOf(e2, payer);
    uint256 borrowerCashPost = sinvoke getCashOf(e2, borrower);
    uint256 otherCashPost = sinvoke getCashOf(e2, other);

    uint256 cTokenBorrowsPost = sinvoke totalBorrows(e2);
    uint256 payerBorrowsPost = sinvoke borrowBalanceStored(e2, payer);
    uint256 borrowerBorrowsPost = sinvoke borrowBalanceStored(e2, borrower);
    uint256 otherBorrowsPost = sinvoke borrowBalanceStored(e2, other);

    uint256 cTokenTokensPost = sinvoke totalSupply(e2);
    uint256 payerTokensPost = sinvoke balanceOf(e2, payer);
    uint256 borrowerTokensPost = sinvoke balanceOf(e2, borrower);
    uint256 otherTokensPost = sinvoke balanceOf(e2, other);

    uint256 cTokenReservesPost = sinvoke totalReserves(e2);

    // Measure
    bool staticBalance =
        (exchangeRatePost == exchangeRatePre) &&
        (cTokenCashPost == cTokenCashPre) &&
        (cTokenBorrowsPost == cTokenBorrowsPre) &&
        (cTokenTokensPost == cTokenTokensPre) &&
        (cTokenReservesPost == cTokenReservesPre) &&
        (payerCashPost == payerCashPre) &&
        (payerBorrowsPost == payerBorrowsPre) &&
        (payerTokensPost == payerTokensPre) &&
        (borrowerCashPost == borrowerCashPre) &&
        (borrowerBorrowsPost == borrowerBorrowsPre) &&
        (borrowerTokensPost == borrowerTokensPre) &&
        (otherCashPost == otherCashPre) &&
        (otherBorrowsPost == otherBorrowsPre) &&
        (otherTokensPost == otherTokensPre);

    // XXX more convenient way to represent uint max?
    uint256 UINT_MAX =
115792089237316195423570985008687907853269984665640564039457584007913129639935;
    uint256 realRepayAmount;
    static_require
        ((repayAmount == UINT_MAX) =>
         (realRepayAmount == borrowerBorrowsPre)) &&
```

```
        ((repayAmount != UINT_MAX) =>
         (realRepayAmount == repayAmount));

    uint256 payerBorrowsExpected;
    uint256 borrowerCashExpected;
    static_require
        ((payer == borrower) =>
         (payerBorrowsExpected == payerBorrowsPre - realRepayAmount) &&
         (borrowerCashExpected == borrowerCashPre - realRepayAmount)) &&
        ((payer != borrower) =>
         (payerBorrowsExpected == payerBorrowsPre) &&
         (borrowerCashExpected == borrowerCashPre));

    bool dynamicBalance =
        (realRepayAmount != 0) &&
        (exchangeRatePost == exchangeRatePre) &&
        (cTokenCashPost == cTokenCashPre + realRepayAmount) &&
        (cTokenBorrowsPost == cTokenBorrowsPre - realRepayAmount) &&
        (cTokenTokensPost == cTokenTokensPre) &&
        (cTokenReservesPost == cTokenReservesPre) &&
        (payerCashPost == payerCashPre - realRepayAmount) &&
        (payerBorrowsPost == payerBorrowsExpected) &&
        (payerTokensPost == payerTokensPre) &&
        (borrowerCashPost == borrowerCashExpected) &&
        (borrowerBorrowsPost == borrowerBorrowsPre - realRepayAmount) &&
        (borrowerTokensPost == borrowerTokensPre) &&
        (otherCashPost == otherCashPre) &&
        (otherBorrowsPost == otherBorrowsPre) &&
        (otherTokensPost == otherTokensPre);

    // XXX would be nice to pull named enums from actual contract?
    static_assert (!repayBorrowFreshReverted =>
                   ((result != 0 || realRepayAmount == 0) <=> staticBalance)), "Mismatch in static
case";
    static_assert (!repayBorrowFreshReverted =>
                   ((result == 0 && realRepayAmount != 0 && payer != currentContract) <=>
dynamicBalance)), "Mismatch in dynamic case";
}
```

Liquidate and Seize

liquidateFresh.cvl

```
methods {
    totalSupply() returns uint256
    totalSupplyInOther() returns uint256 // not part of API
    totalBorrows() returns uint256
    totalBorrowsInOther() returns uint256 // not part of API
    totalReserves() returns uint256
    totalReservesInOther() returns uint256 // not part of API

    balanceOf(address) returns uint256
    balanceOfInOther(address) returns uint256 // not part of API
    borrowBalanceStored(address) returns uint256
    borrowBalanceStoredInOther(address) returns uint256 // not part of API
    exchangeRateStored() returns uint256
    exchangeRateStoredInOther() returns uint256 // not part of API

    getCash() returns uint256
    getCashInOther() returns uint256 // not part of API
    getCashOf(address) returns uint256 // not part of API
    getCashOfInOther(address) returns uint256 // not part of API

    liquidateBorrowFreshPub(address, address, uint256) returns uint256
    seize(address, address, uint256) returns uint256
}

rule liquidateBorrowFresh(uint result, address liquidator, address borrower, uint256 repayAmount)
description "Break liquidate" {
    // Pre/action/post environments
    env e0; havoc e0;
    env e1; havoc e1;
    env e2; havoc e2;

    static_require e1.block.number >= e0.block.number;
    static_require e2.block.number >= e1.block.number;

    // Any other account
    address other; havoc other;
    static_require other != liquidator && other != borrower && other != currentContract;

    // We assume this cannot happen
    //  OQ: should we enforce in the code such that it need not be assumed?
    static_require borrower != currentContract;
```

```
/*
  - no effect on exchange rate
  - self-liquidate has no effect
  |--------------------+--------+------------+----------+-------|
  |                    | CToken | Liquidator | Borrower | Other |
  |--------------------+--------+------------+----------+-------|
  | cash               |   +A   |     -A     |     0    |   0   |
  | borrows            |   -A   |      0     |    -A    |   0   |
  | tokens             |    0   |      0     |     0    |   0   |
  | reserves           |    0   |            |          |       |
  | collateral cash    |    0   |      0     |     0    |   0   |
  | collateral borrows |    0   |      0     |     0    |   0   |
  | collateral tokens  |    0   |     +T     |    -T    |   0   |
  | collateral reserves|    0   |            |          |       |
  |--------------------+--------+------------+----------+-------|
*/

/* Pre */

// borrowed
uint256 exchangeRatePre = sinvoke exchangeRateStored(e0);

uint256 cTokenCashPre = sinvoke getCash(e0);
uint256 liquidatorCashPre = sinvoke getCashOf(e0, liquidator);
uint256 borrowerCashPre = sinvoke getCashOf(e0, borrower);
uint256 otherCashPre = sinvoke getCashOf(e0, other);

uint256 cTokenBorrowsPre = sinvoke totalBorrows(e0);
uint256 liquidatorBorrowsPre = sinvoke borrowBalanceStored(e0, liquidator);
uint256 borrowerBorrowsPre = sinvoke borrowBalanceStored(e0, borrower);
uint256 otherBorrowsPre = sinvoke borrowBalanceStored(e0, other);

uint256 cTokenTokensPre = sinvoke totalSupply(e0);
uint256 liquidatorTokensPre = sinvoke balanceOf(e0, liquidator);
uint256 borrowerTokensPre = sinvoke balanceOf(e0, borrower);
uint256 otherTokensPre = sinvoke balanceOf(e0, other);

uint256 cTokenReservesPre = sinvoke totalReserves(e0);

// collateral
uint256 collateralExchangeRatePre = sinvoke exchangeRateStoredInOther(e0);

uint256 collateralCashPre = sinvoke getCashInOther(e0);
uint256 liquidatorCollateralCashPre = sinvoke getCashOfInOther(e0, liquidator);
uint256 borrowerCollateralCashPre = sinvoke getCashOfInOther(e0, borrower);
uint256 otherCollateralCashPre = sinvoke getCashOfInOther(e0, other);
```

```
    uint256 collateralBorrowsPre = sinvoke totalBorrowsInOther(e0);
    uint256 liquidatorCollateralBorrowsPre = sinvoke borrowBalanceStoredInOther(e0, liquidator);
    uint256 borrowerCollateralBorrowsPre = sinvoke borrowBalanceStoredInOther(e0, borrower);
    uint256 otherCollateralBorrowsPre = sinvoke borrowBalanceStoredInOther(e0, other);

    uint256 collateralTokensPre = sinvoke totalSupplyInOther(e0);
    uint256 liquidatorCollateralTokensPre = sinvoke balanceOfInOther(e0, liquidator);
    uint256 borrowerCollateralTokensPre = sinvoke balanceOfInOther(e0, borrower);
    uint256 otherCollateralTokensPre = sinvoke balanceOfInOther(e0, other);

    uint256 collateralReservesPre = sinvoke totalReservesInOther(e0);

    // Just Do It
    // Note: cTokenCollateral is linked via Compound.spclnk in order to support checking its
balances
    //  not perfect since it only proves the balance sheet is safe for a particular token
configuration
    static_require result == invoke liquidateBorrowFreshPub(e1, liquidator, borrower, repayAmount);
    bool liquidateBorrowFreshReverted = lastReverted;

    /* Post */

    // borrowed
    uint256 exchangeRatePost = sinvoke exchangeRateStored(e2);

    uint256 cTokenCashPost = sinvoke getCash(e2);
    uint256 liquidatorCashPost = sinvoke getCashOf(e2, liquidator);
    uint256 borrowerCashPost = sinvoke getCashOf(e2, borrower);
    uint256 otherCashPost = sinvoke getCashOf(e2, other);

    uint256 cTokenBorrowsPost = sinvoke totalBorrows(e2);
    uint256 liquidatorBorrowsPost = sinvoke borrowBalanceStored(e2, liquidator);
    uint256 borrowerBorrowsPost = sinvoke borrowBalanceStored(e2, borrower);
    uint256 otherBorrowsPost = sinvoke borrowBalanceStored(e2, other);

    uint256 cTokenTokensPost = sinvoke totalSupply(e2);
    uint256 liquidatorTokensPost = sinvoke balanceOf(e2, liquidator);
    uint256 borrowerTokensPost = sinvoke balanceOf(e2, borrower);
    uint256 otherTokensPost = sinvoke balanceOf(e2, other);

    uint256 cTokenReservesPost = sinvoke totalReserves(e2);

    // collateral
    uint256 collateralExchangeRatePost = sinvoke exchangeRateStoredInOther(e2);
```

```
uint256 collateralCashPost = sinvoke getCashInOther(e2);
uint256 liquidatorCollateralCashPost = sinvoke getCashOfInOther(e2, liquidator);
uint256 borrowerCollateralCashPost = sinvoke getCashOfInOther(e2, borrower);
uint256 otherCollateralCashPost = sinvoke getCashOfInOther(e2, other);

uint256 collateralBorrowsPost = sinvoke totalBorrowsInOther(e2);
uint256 liquidatorCollateralBorrowsPost = sinvoke borrowBalanceStoredInOther(e2, liquidator);
uint256 borrowerCollateralBorrowsPost = sinvoke borrowBalanceStoredInOther(e2, borrower);
uint256 otherCollateralBorrowsPost = sinvoke borrowBalanceStoredInOther(e2, other);

uint256 collateralTokensPost = sinvoke totalSupplyInOther(e2);
uint256 liquidatorCollateralTokensPost = sinvoke balanceOfInOther(e2, liquidator);
uint256 borrowerCollateralTokensPost = sinvoke balanceOfInOther(e2, borrower);
uint256 otherCollateralTokensPost = sinvoke balanceOfInOther(e2, other);

uint256 collateralReservesPost = sinvoke totalReservesInOther(e2);

// Measure
bool staticBalance =
    (exchangeRatePost == exchangeRatePre) &&
    (cTokenCashPost == cTokenCashPre) &&
    (cTokenBorrowsPost == cTokenBorrowsPre) &&
    (cTokenTokensPost == cTokenTokensPre) &&
    (cTokenReservesPost == cTokenReservesPre) &&
    (liquidatorCashPost == liquidatorCashPre) &&
    (liquidatorBorrowsPost == liquidatorBorrowsPre) &&
    (liquidatorTokensPost == liquidatorTokensPre) &&
    (borrowerCashPost == borrowerCashPre) &&
    (borrowerBorrowsPost == borrowerBorrowsPre) &&
    (borrowerTokensPost == borrowerTokensPre) &&
    (otherCashPost == otherCashPre) &&
    (otherBorrowsPost == otherBorrowsPre) &&
    (otherTokensPost == otherTokensPre) &&

    (collateralExchangeRatePost == collateralExchangeRatePre) &&
    (collateralCashPost == collateralCashPre) &&
    (collateralBorrowsPost == collateralBorrowsPre) &&
    (collateralTokensPost == collateralTokensPre) &&
    (collateralReservesPost == collateralReservesPre) &&
    (liquidatorCollateralCashPost == liquidatorCollateralCashPre) &&
    (liquidatorCollateralBorrowsPost == liquidatorCollateralBorrowsPre) &&
    (liquidatorCollateralTokensPost == liquidatorCollateralTokensPre) &&
    (borrowerCollateralCashPost == borrowerCollateralCashPre) &&
    (borrowerCollateralBorrowsPost == borrowerCollateralBorrowsPre) &&
    (borrowerCollateralTokensPost == borrowerCollateralTokensPre) &&
    (otherCollateralCashPost == otherCollateralCashPre) &&
```

```
        (otherCollateralBorrowsPost == otherCollateralBorrowsPre) &&
        (otherCollateralTokensPost == otherCollateralTokensPre);

    bool dynamicBalance =
        (repayAmount != 0) &&
        (exchangeRatePost == exchangeRatePre) &&
        (cTokenCashPost == cTokenCashPre + repayAmount) &&
        (cTokenBorrowsPost == cTokenBorrowsPre - repayAmount) &&
        (cTokenTokensPost == cTokenTokensPre) &&
        (cTokenReservesPost == cTokenReservesPre) &&
        (liquidatorCashPost == liquidatorCashPre - repayAmount) &&
        (liquidatorBorrowsPost == liquidatorBorrowsPre) &&
        (liquidatorTokensPost == liquidatorTokensPre) &&
        (borrowerCashPost == borrowerCashPre) &&
        (borrowerBorrowsPost == borrowerBorrowsPre - repayAmount) &&
        (borrowerTokensPost == borrowerTokensPre) &&
        (otherCashPost == otherCashPre) &&
        (otherBorrowsPost == otherBorrowsPre) &&
        (otherTokensPost == otherTokensPre) &&
        (borrowerCollateralCashPost == borrowerCollateralCashPre) &&

        (collateralExchangeRatePost == collateralExchangeRatePre) &&
        (collateralCashPost == collateralCashPre) &&
        (collateralBorrowsPost == collateralBorrowsPre) &&
        (collateralTokensPost == collateralTokensPre) &&
        (collateralReservesPost == collateralReservesPre) &&
        (liquidatorCollateralCashPost == liquidatorCollateralCashPre) &&
        (liquidatorCollateralBorrowsPost == liquidatorCollateralBorrowsPre) &&
        (liquidatorCollateralTokensPost - liquidatorCollateralTokensPre ==
borrowerCollateralTokensPre - borrowerCollateralTokensPost) &&
        (borrowerCollateralCashPost == borrowerCollateralCashPre) &&
        (borrowerCollateralBorrowsPost == borrowerCollateralBorrowsPre) &&
        (otherCollateralCashPost == otherCollateralCashPre) &&
        (otherCollateralBorrowsPost == otherCollateralBorrowsPre) &&
        (otherCollateralTokensPost == otherCollateralTokensPre);


    static_assert (!liquidateBorrowFreshReverted =>
                   ((result != 0 || repayAmount == 0 || liquidator == borrower) <=> staticBalance)),
"Mismatch in static case";
    static_assert (!liquidateBorrowFreshReverted =>
                   ((result == 0 && repayAmount != 0 && liquidator != currentContract) <=>
dynamicBalance)), "Mismatch in dynamic case";
}

rule seize(uint result, address liquidator, address borrower, uint256 seizeTokens)
```

```
description "Break seize" {
    // Pre/action/post environments
    env e0; havoc e0;
    env e1; havoc e1;
    env e2; havoc e2;

    static_require e1.block.number >= e0.block.number;
    static_require e2.block.number >= e1.block.number;

    // Any other account
    address other; havoc other;
    static_require other != liquidator && other != borrower && other != currentContract;

    /*
      - no effect on exchange rate
      |----------+--------+------------+----------+-------|
      |          | CToken | Liquidator | Borrower | Other |
      |----------+--------+------------+----------+-------|
      | cash     |    0   |      0     |     0    |   0   |
      | borrows  |    0   |      0     |     0    |   0   |
      | tokens   |    0   |     +T     |    -T    |   0   |
      | reserves |    0   |            |          |       |
      |----------+--------+------------+----------+-------|
    */

    /* Pre */

    uint256 exchangeRatePre = sinvoke exchangeRateStored(e0);

    uint256 cTokenCashPre = sinvoke getCash(e0);
    uint256 liquidatorCashPre = sinvoke getCashOf(e0, liquidator);
    uint256 borrowerCashPre = sinvoke getCashOf(e0, borrower);
    uint256 otherCashPre = sinvoke getCashOf(e0, other);

    uint256 cTokenBorrowsPre = sinvoke totalBorrows(e0);
    uint256 liquidatorBorrowsPre = sinvoke borrowBalanceStored(e0, liquidator);
    uint256 borrowerBorrowsPre = sinvoke borrowBalanceStored(e0, borrower);
    uint256 otherBorrowsPre = sinvoke borrowBalanceStored(e0, other);

    uint256 cTokenTokensPre = sinvoke totalSupply(e0);
    uint256 liquidatorTokensPre = sinvoke balanceOf(e0, liquidator);
    uint256 borrowerTokensPre = sinvoke balanceOf(e0, borrower);
    uint256 otherTokensPre = sinvoke balanceOf(e0, other);

    uint256 cTokenReservesPre = sinvoke totalReserves(e0);
```

```
// Just Do It
static_require result == invoke seize(e1, liquidator, borrower, seizeTokens);
bool seizeReverted = lastReverted;

/* Post */
uint256 exchangeRatePost = sinvoke exchangeRateStored(e2);

uint256 cTokenCashPost = sinvoke getCash(e2);
uint256 liquidatorCashPost = sinvoke getCashOf(e2, liquidator);
uint256 borrowerCashPost = sinvoke getCashOf(e2, borrower);
uint256 otherCashPost = sinvoke getCashOf(e2, other);

uint256 cTokenBorrowsPost = sinvoke totalBorrows(e2);
uint256 liquidatorBorrowsPost = sinvoke borrowBalanceStored(e2, liquidator);
uint256 borrowerBorrowsPost = sinvoke borrowBalanceStored(e2, borrower);
uint256 otherBorrowsPost = sinvoke borrowBalanceStored(e2, other);

uint256 cTokenTokensPost = sinvoke totalSupply(e2);
uint256 liquidatorTokensPost = sinvoke balanceOf(e2, liquidator);
uint256 borrowerTokensPost = sinvoke balanceOf(e2, borrower);
uint256 otherTokensPost = sinvoke balanceOf(e2, other);

uint256 cTokenReservesPost = sinvoke totalReserves(e2);

// Measure
bool staticBalance =
    (exchangeRatePost == exchangeRatePre) &&
    (cTokenCashPost == cTokenCashPre) &&
    (cTokenBorrowsPost == cTokenBorrowsPre) &&
    (cTokenTokensPost == cTokenTokensPre) &&
    (cTokenReservesPost == cTokenReservesPre) &&
    (liquidatorCashPost == liquidatorCashPre) &&
    (liquidatorBorrowsPost == liquidatorBorrowsPre) &&
    (liquidatorTokensPost == liquidatorTokensPre) &&
    (borrowerCashPost == borrowerCashPre) &&
    (borrowerBorrowsPost == borrowerBorrowsPre) &&
    (borrowerTokensPost == borrowerTokensPre) &&
    (otherCashPost == otherCashPre) &&
    (otherBorrowsPost == otherBorrowsPre) &&
    (otherTokensPost == otherTokensPre);

bool dynamicBalance =
    (seizeTokens != 0) &&
    (exchangeRatePost == exchangeRatePre) &&
    (cTokenCashPost == cTokenCashPre) &&
    (cTokenBorrowsPost == cTokenBorrowsPre) &&
```

```
        (cTokenTokensPost == cTokenTokensPre) &&
        (cTokenReservesPost == cTokenReservesPre) &&
        (liquidatorCashPost == liquidatorCashPre) &&
        (liquidatorBorrowsPost == liquidatorBorrowsPre) &&
        (liquidatorTokensPost == liquidatorTokensPre + seizeTokens) &&
        (borrowerCashPost == borrowerCashPre) &&
        (borrowerBorrowsPost == borrowerBorrowsPre) &&
        (borrowerTokensPost == borrowerTokensPre - seizeTokens) &&
        (otherCashPost == otherCashPre) &&
        (otherBorrowsPost == otherBorrowsPre) &&
        (otherTokensPost == otherTokensPre);

    static_assert (!seizeReverted =>
                   ((result != 0 || seizeTokens == 0 || liquidator == borrower) <=> staticBalance)),
"Mismatch in static case";
    static_assert (!seizeReverted =>
                   ((result == 0 && seizeTokens != 0) <=> dynamicBalance)), "Mismatch in dynamic
case";
}
```

ERC20 functionality

EIP20.cvl

```
methods {
    totalSupply() returns uint256
    totalBorrows() returns uint256
    totalReserves() returns uint256

    balanceOf(address) returns uint256
    borrowBalanceStored(address) returns uint256
    exchangeRateStored() returns uint256

    getCash() returns uint256
    getCashOf(address) returns uint256 // not part of API

    allowance(address, address) returns uint256
    approve(address, uint256) returns bool
    transferFrom(address, address, uint256) returns bool
}

rule transferFrom(bool success, address src, address dst, uint256 amount)
description "Break transferFrom" {
    // Pre/action/post environments
    env e0; havoc e0;
    env e1a; havoc e1a;
    env e1b; havoc e1b;
    env e2; havoc e2;

    static_require e1a.block.number >= e0.block.number;
    static_require e1b.block.number >= e1a.block.number;
    static_require e2.block.number >= e1b.block.number;

    // Any other account
    address other; havoc other;
    static_require other != src && other != dst;

    /*
      - no effect on exchange rate
      - no more than approved
      |----------+--------+-----+-----+-------|
      |          | CToken | Src | Dst | Other |
      |----------+--------+-----+-----+-------|
      | cash     |    0   |  0  |  0  |   0   |
      | borrows  |    0   |  0  |  0  |   0   |
      | tokens   |    0   | -T  | +T  |   0   |
```

```
  | reserves |    0   |     |     |       |
  |----------+--------+-----+-----+-------|
*/

/* Pre */

uint256 exchangeRatePre = sinvoke exchangeRateStored(e0);

uint256 cTokenCashPre = sinvoke getCash(e0);
uint256 srcCashPre = sinvoke getCashOf(e0, src);
uint256 dstCashPre = sinvoke getCashOf(e0, dst);
uint256 otherCashPre = sinvoke getCashOf(e0, other);

uint256 cTokenBorrowsPre = sinvoke totalBorrows(e0);
uint256 srcBorrowsPre = sinvoke borrowBalanceStored(e0, src);
uint256 dstBorrowsPre = sinvoke borrowBalanceStored(e0, dst);
uint256 otherBorrowsPre = sinvoke borrowBalanceStored(e0, other);

uint256 cTokenSupplyPre = sinvoke totalSupply(e0);
uint256 srcTokensPre = sinvoke balanceOf(e0, src);
uint256 dstTokensPre = sinvoke balanceOf(e0, dst);
uint256 otherTokensPre = sinvoke balanceOf(e0, other);

uint256 cTokenReservesPre = sinvoke totalReserves(e0);

// Approve
bool doApprove; havoc doApprove;
uint256 approvedAmount; havoc approvedAmount;
if (doApprove) {
    static_require e1a.msg.sender == src;
    sinvoke approve(e1a, e1b.msg.sender, approvedAmount);
} else {}

uint256 allowancePre = sinvoke allowance(e1a, src, e1b.msg.sender);

// Just Do It
static_require success == invoke transferFrom(e1b, src, dst, amount);
bool transferReverted = lastReverted;

/* Post */

uint256 exchangeRatePost = sinvoke exchangeRateStored(e2);

uint256 cTokenCashPost = sinvoke getCash(e2);
uint256 srcCashPost = sinvoke getCashOf(e2, src);
uint256 dstCashPost = sinvoke getCashOf(e2, dst);
```

```
    uint256 otherCashPost = sinvoke getCashOf(e2, other);

    uint256 cTokenBorrowsPost = sinvoke totalBorrows(e2);
    uint256 srcBorrowsPost = sinvoke borrowBalanceStored(e2, src);
    uint256 dstBorrowsPost = sinvoke borrowBalanceStored(e2, dst);
    uint256 otherBorrowsPost = sinvoke borrowBalanceStored(e2, other);

    uint256 cTokenSupplyPost = sinvoke totalSupply(e2);
    uint256 srcTokensPost = sinvoke balanceOf(e2, src);
    uint256 dstTokensPost = sinvoke balanceOf(e2, dst);
    uint256 otherTokensPost = sinvoke balanceOf(e2, other);

    uint256 cTokenReservesPost = sinvoke totalReserves(e2);

    uint256 allowancePost = sinvoke allowance(e2, src, e1b.msg.sender);

    // Measure
    bool staticBalance =
        (exchangeRatePost == exchangeRatePre) &&
        (cTokenCashPost == cTokenCashPre) &&
        (cTokenBorrowsPost == cTokenBorrowsPre) &&
        (cTokenSupplyPost == cTokenSupplyPre) &&
        (cTokenReservesPost == cTokenReservesPre) &&
        (srcCashPost == srcCashPre) &&
        (srcBorrowsPost == srcBorrowsPre) &&
        (srcTokensPost == srcTokensPre) &&
        (dstCashPost == dstCashPre) &&
        (dstBorrowsPost == dstBorrowsPre) &&
        (dstTokensPost == dstTokensPre) &&
        (otherCashPost == otherCashPre) &&
        (otherBorrowsPost == otherBorrowsPre) &&
        (otherTokensPost == otherTokensPre);

    bool dynamicBalance =
        (amount != 0) &&
        (exchangeRatePost == exchangeRatePre) &&
        (cTokenCashPost == cTokenCashPre) &&
        (cTokenBorrowsPost == cTokenBorrowsPre) &&
        (cTokenSupplyPost == cTokenSupplyPre) &&
        (cTokenReservesPost == cTokenReservesPre) &&
        (srcCashPost == srcCashPre) &&
        (srcBorrowsPost == srcBorrowsPre) &&
        (srcTokensPost == srcTokensPre - amount) &&
        (dstCashPost == dstCashPre) &&
        (dstBorrowsPost == dstBorrowsPre) &&
        (dstTokensPost == dstTokensPre + amount) &&
```

```
        (otherCashPost == otherCashPre) &&
        (otherBorrowsPost == otherBorrowsPre) &&
        (otherTokensPost == otherTokensPre);

    // XXX better way to write uint max?
    uint256 UINT_MAX =
115792089237316195423570985008687907853269984665640564039457584007913129639935;

    static_assert (!transferReverted =>
                   ((!success || amount == 0 || src == dst) <=> staticBalance)), "Mismatch in static
case";
    static_assert (!transferReverted =>
                   ((success && amount != 0) <=> dynamicBalance)), "Mismatch in dynamic case";
    static_assert (!transferReverted && success =>
                   (amount > allowancePre => e1b.msg.sender == src)), "Only owner can transfer >
allowance";
    static_assert (!transferReverted && success =>
                   (doApprove => allowancePre >= approvedAmount)), "Approval must increase the
allowance";
    static_assert (!transferReverted && success =>
                   (allowancePre == UINT_MAX || e1b.msg.sender == src || amount == 0) <=>
allowancePost == allowancePre), "Mismatch not touching allowance";
    static_assert (!transferReverted && success && e1b.msg.sender != src && amount != 0 =>
                   (allowancePre != UINT_MAX <=> allowancePost == allowancePre - amount)), "Spender
transfer uses allowance";
}
```

Frame properties - mapping out state changes

Link to results:
https://docs.google.com/spreadsheets/d/1NaXIci4vrxtBgJoOzVObdZHEmhA4a_iIovZr954KkKc/edit?usp=sharing

frame.cvl

```
methods {
/*70a08231:*/ balanceOf(address)                  returns uint256
/*95dd9193:*/ borrowBalanceStored(address)  returns uint256
/*aa5af0fd:*/ borrowIndex()                 returns uint256
/*f8f9da28:*/ borrowRatePerBlock()          returns uint256
/*5fe3b567:*/ comptroller()                 returns address
/*182df0f5:*/ exchangeRateStored()          returns uint256
/*c37f68e2:*/ getAccountSnapshot(address)   returns uint256,uint256,uint256,uint256
/*3b1d21a2:*/ getCash()                     returns uint256
/*675d972c:*/ initialExchangeRateMantissa() returns uint256
/*f3fdb15a:*/ interestRateModel()           returns address
/*fe9c44ae:*/ isCToken()                    returns bool
/*26782247:*/ pendingAdmin()                returns address
/*173b9904:*/ reserveFactorMantissa()       returns uint256
/*ae9d70b0:*/ supplyRatePerBlock()          returns uint256
/*47bd3718:*/ totalBorrows()                returns uint256
/*8f840ddd:*/ totalReserves()               returns uint256
/*18160ddd:*/ totalSupply()                 returns uint256


// all other methods
/*e9c714f2:*/ _acceptAdmin()
/*601a0bf1:*/ _reduceReserves(uint256)
/*4576b5db:*/ _setComptroller(address)
/*f2b3abbd:*/ _setInterestRateModel(address)
/*b71d1a0c:*/ _setPendingAdmin(address)
/*fca7820b:*/ _setReserveFactor(uint256)
/*6c540baf:*/ accrualBlockNumber()
/*a6afed95:*/ accrueInterest()
/*f851a440:*/ admin()
/*dd62ed3e:*/ allowance(address,address)
/*095ea7b3:*/ approve(address,uint256)
/*3af9e669:*/ balanceOfUnderlying(address)
/*c5ebeaec:*/ borrow(uint256)
/*17bfdfbc:*/ borrowBalanceCurrent(address)
/*0bc1d628:*/ borrowFreshPub(address,uint256)
/*5d2b2256:*/ cTokenMintComputation(uint256)
/*f1651460:*/ cTokenRedeemComputation(uint256)
/*b8b8b26b:*/ checkTransferInPub(address,uint256)
```

```
/*d9d9e5e5:*/ comptrollerMintAllowed(address,address,uint256)
/*92ac96fa:*/ comptrollerRedeemAllowed(address,address,uint256)
/*313ce567:*/ decimals()
/*19ec23ba:*/ doTransferInPub(address,uint256)
/*f782def5:*/ doTransferInPubSim(address,uint256)
/*b5d229bb:*/ doTransferOutPub(address,uint256)
/*18628ce7:*/ doTransferOutPubSim(address,uint256)
/*bd6d894d:*/ exchangeRateCurrent()
/*934213e8:*/ exchangeRateStoredInternalPub()
/*f414bbaf:*/ getCashOf(address)
/*5b1c10f6:*/ getCashPub()
/*529ca4b7:*/ interestRateModelGetBorrowRate()
/*f5e3c462:*/ liquidateBorrow(address,uint256,address)
/*a0712d68:*/ mint(uint256)
/*c9645990:*/ mintFreshPub(address,uint256)
/*06fdde03:*/ name()
/*db006a75:*/ redeem(uint256)
/*8c1a1b80:*/ redeemFreshPub(address,uint256,uint256)
/*852a12e3:*/ redeemUnderlying(uint256)
/*0e752702:*/ repayBorrow(uint256)
/*2608f818:*/ repayBorrowBehalf(address,uint256)
/*7a6a0162:*/ repayBorrowFreshPub(address,address,uint256)
/*b2a02ff1:*/ seize(address,address,uint256)
/*95d89b41:*/ symbol()
/*73acee98:*/ totalBorrowsCurrent()
/*a9059cbb:*/ transfer(address,uint256)
/*23b872dd:*/ transferFrom(address,address,uint256)
/*6f307dc3:*/ underlying()
}

rule frame_balanceOf(address a, method f)
description "$f may change value of balanceOf($a)"
{
        env e0; havoc e0;
        env e1; havoc e1;
        env e2; havoc e2;
        static_require e1.block.number >= e0.block.number;
        static_require e2.block.number >= e1.block.number;

        uint256 old = sinvoke balanceOf(e0, a);
        calldataarg arg;
        invoke f(e1, arg);
        uint256 new = sinvoke balanceOf(e2, a);

        static_assert old == new;
}
```

```
rule frame_borrowBalanceStored(address a, method f)
description "$f may change value of borrowBalanceStored($a)"
{
        env e0; havoc e0;
        env e1; havoc e1;
        env e2; havoc e2;
        static_require e1.block.number >= e0.block.number;
        static_require e2.block.number >= e1.block.number;

        uint256 old = sinvoke borrowBalanceStored(e0, a);
        calldataarg arg;
        invoke f(e1, arg);
        uint256 new = sinvoke borrowBalanceStored(e2, a);

        static_assert old == new;
}

rule frame_borrowIndex(method f)
description "$f may change value of borrowIndex()"
{
        env e0; havoc e0;
        env e1; havoc e1;
        env e2; havoc e2;
        static_require e1.block.number >= e0.block.number;
        static_require e2.block.number >= e1.block.number;

        uint256 old = sinvoke borrowIndex(e0);
        calldataarg arg;
        invoke f(e1, arg);
        uint256 new = sinvoke borrowIndex(e2);

        static_assert old == new;
}

rule frame_borrowRatePerBlock(method f)
description "$f may change value of borrowRatePerBlock()"
{
        env e0; havoc e0;
        env e1; havoc e1;
        env e2; havoc e2;
        static_require e1.block.number >= e0.block.number;
        static_require e2.block.number >= e1.block.number;

        uint256 old = sinvoke borrowRatePerBlock(e0);
        calldataarg arg;
```

```
        invoke f(e1, arg);
        uint256 new = sinvoke borrowRatePerBlock(e2);

        static_assert old == new;
}

rule frame_comptroller(method f)
description "$f may change value of comptroller()"
{
        env e0; havoc e0;
        env e1; havoc e1;
        env e2; havoc e2;
        static_require e1.block.number >= e0.block.number;
        static_require e2.block.number >= e1.block.number;

        address old = sinvoke comptroller(e0);
        calldataarg arg;
        invoke f(e1, arg);
        address new = sinvoke comptroller(e2);

        static_assert old == new;
}

rule frame_exchangeRateStored(address a, method f)
description "$f may change value of exchangeRateStored()"
{
        env e0; havoc e0;
        env e1; havoc e1;
        env e2; havoc e2;
        static_require e1.block.number >= e0.block.number;
        static_require e2.block.number >= e1.block.number;

        uint256 old = sinvoke exchangeRateStored(e0);
        calldataarg arg;
        invoke f(e1, arg);
        uint256 new = sinvoke exchangeRateStored(e2);

        static_assert old == new;
}

rule frame_getAccountSnapshot(address a, method f)
description "$f may change value of getAccountSnapshot($a)"
{
        env e0; havoc e0;
        env e1; havoc e1;
        env e2; havoc e2;
```

```
        static_require e1.block.number >= e0.block.number;
        static_require e2.block.number >= e1.block.number;

        uint256 old1; uint256 old2; uint256 old3; uint256 old4;
        old1,old2,old3,old4 = sinvoke getAccountSnapshot(e0, a);
        calldataarg arg;
        invoke f(e1, arg);
        uint256 new1; uint256 new2; uint256 new3; uint256 new4;
        new1,new2,new3,new4 = sinvoke getAccountSnapshot(e2, a);

        static_assert old1 == new1 && old2 == new2 && old3 == new3 && old4 == new4;
}

rule frame_getCash(method f)
description "$f may change value of getCash()"
{
        env e0; havoc e0;
        env e1; havoc e1;
        env e2; havoc e2;
        static_require e1.block.number >= e0.block.number;
        static_require e2.block.number >= e1.block.number;

        uint256 old = sinvoke getCash(e0);
        calldataarg arg;
        invoke f(e1, arg);
        uint256 new = sinvoke getCash(e2);

        static_assert old == new;
}

rule frame_initialExchangeRateMantissa(method f)
description "$f may change value of initialExchangeRateMantissa()"
{
        env e0; havoc e0;
        env e1; havoc e1;
        env e2; havoc e2;
        static_require e1.block.number >= e0.block.number;
        static_require e2.block.number >= e1.block.number;

        uint256 old = sinvoke initialExchangeRateMantissa(e0);
        calldataarg arg;
        invoke f(e1, arg);
        uint256 new = sinvoke initialExchangeRateMantissa(e2);

        static_assert old == new;
}
```

```
rule frame_interestRateModel(method f)
description "$f may change value of interestRateModel()"
{
        env e0; havoc e0;
        env e1; havoc e1;
        env e2; havoc e2;
        static_require e1.block.number >= e0.block.number;
        static_require e2.block.number >= e1.block.number;

        address old = sinvoke interestRateModel(e0);
        calldataarg arg;
        invoke f(e1, arg);
        address new = sinvoke interestRateModel(e2);

        static_assert old == new;
}

rule frame_isCToken(method f)
description "$f may change value of isCToken()"
{
        env e0; havoc e0;
        env e1; havoc e1;
        env e2; havoc e2;
        static_require e1.block.number >= e0.block.number;
        static_require e2.block.number >= e1.block.number;

        bool old = sinvoke isCToken(e0);
        calldataarg arg;
        invoke f(e1, arg);
        bool new = sinvoke isCToken(e2);

        static_assert old == new;
}

rule frame_pendingAdmin(method f)
description "$f may change value of pendingAdmin()"
{
        env e0; havoc e0;
        env e1; havoc e1;
        env e2; havoc e2;
        static_require e1.block.number >= e0.block.number;
        static_require e2.block.number >= e1.block.number;

        address old = sinvoke pendingAdmin(e0);
        calldataarg arg;
```

```
        invoke f(e1, arg);
        address new = sinvoke pendingAdmin(e2);

        static_assert old == new;
}

rule frame_reserveFactorMantissa(method f)
description "$f may change value of reserveFactorMantissa()"
{
        env e0; havoc e0;
        env e1; havoc e1;
        env e2; havoc e2;
        static_require e1.block.number >= e0.block.number;
        static_require e2.block.number >= e1.block.number;

        uint256 old = sinvoke reserveFactorMantissa(e0);
        calldataarg arg;
        invoke f(e1, arg);
        uint256 new = sinvoke reserveFactorMantissa(e2);

        static_assert old == new;
}

rule frame_supplyRatePerBlock(method f)
description "$f may change value of supplyRatePerBlock()"
{
        env e0; havoc e0;
        env e1; havoc e1;
        env e2; havoc e2;
        static_require e1.block.number >= e0.block.number;
        static_require e2.block.number >= e1.block.number;

        uint256 old = sinvoke supplyRatePerBlock(e0);
        calldataarg arg;
        invoke f(e1, arg);
        uint256 new = sinvoke supplyRatePerBlock(e2);

        static_assert old == new;
}

rule frame_totalBorrows(method f)
description "$f may change value of totalBorrows()"
{
        env e0; havoc e0;
        env e1; havoc e1;
        env e2; havoc e2;
```

```
        static_require e1.block.number >= e0.block.number;
        static_require e2.block.number >= e1.block.number;

        uint256 old = sinvoke totalBorrows(e0);
        calldataarg arg;
        invoke f(e1, arg);
        uint256 new = sinvoke totalBorrows(e2);

        static_assert old == new;
}

rule frame_totalReserves(address a, method f)
description "$f may change value of totalReserves()"
{
        env e0; havoc e0;
        env e1; havoc e1;
        env e2; havoc e2;
        static_require e1.block.number >= e0.block.number;
        static_require e2.block.number >= e1.block.number;

        uint256 old = sinvoke totalReserves(e0);
        calldataarg arg;
        invoke f(e1, arg);
        uint256 new = sinvoke totalReserves(e2);

        static_assert old == new;
}

rule frame_totalSupply(method f)
description "$f may change value of totalSupply()"
{
        env e0; havoc e0;
        env e1; havoc e1;
        env e2; havoc e2;
        static_require e1.block.number >= e0.block.number;
        static_require e2.block.number >= e1.block.number;

        uint256 old = sinvoke totalSupply(e0);
        calldataarg arg;
        invoke f(e1, arg);
        uint256 new = sinvoke totalSupply(e2);

        static_assert old == new;
}
```

Unitroller

Administrative functions

general.cvl

```
methods {
        _setPendingAdmin(address) returns uint256
        pendingAdmin() returns address

        _acceptAdmin() returns uint256
        admin() returns address

       _setPendingImplementation(address) returns uint256
        _acceptImplementation(address) returns uint256
        comptrollerImplementation() returns address
       pendingComptrollerImplementation() returns address
}

rule _setPendingAdmin(address currentAdmin, address currentPendingAdmin, address newPendingAdmin)
description "Failed to set new pending admin $currentPendingAdmin to $newPendingAdmin
(admin=$currentAdmin)"
{
        // Free Variables
        env e0; havoc e0;
        env e1; havoc e1;
        env e2; havoc e2;

        // Strict ordering
        static_require e1.block > e0.block;
        static_require e2.block > e1.block;

        static_require currentAdmin == sinvoke admin(e0);
        static_require currentPendingAdmin == sinvoke pendingAdmin(e0);

        // Invoke set new pending admin
        uint256 result = sinvoke _setPendingAdmin(e1, newPendingAdmin);

        // pendingAdmin changes <=> msg.sender == currentAdmin
        static_assert (
                        (
                        e1.msg.sender == currentAdmin &&
                        result == 0 &&
                        sinvoke pendingAdmin(e2) == newPendingAdmin
                        )
```

```
                    ||
                    (
                    e1.msg.sender != currentAdmin &&
                    result != 0 &&
                    sinvoke pendingAdmin(e2) == currentPendingAdmin
                    )
                    );
}

rule _acceptAdmin(address currentAdmin, address currentPendingAdmin, address newAdmin, address
newPendingAdmin)
        description "Failed to accept pending admin currentAdmin=$currentAdmin,
currentPendingAdmin=$currentPendingAdmin, newPendingAdmin=$newPendingAdmin, newAdmin=$newAdmin"
{
        // Free Variables
        env e0; havoc e0;
        env e1; havoc e1;
        env e2; havoc e2;

        // Strict ordering
        static_require e1.block > e0.block;
        static_require e2.block > e1.block;

        static_require currentAdmin == sinvoke admin(e0);
        static_require currentPendingAdmin == sinvoke pendingAdmin(e0);

        // Invoke accept admin
        uint256 result = sinvoke _acceptAdmin(e1);

        static_require newAdmin == sinvoke admin(e2);
        static_require newPendingAdmin == sinvoke pendingAdmin(e2);

        // admin == pendingAdmin <=> msg.sender == pendingAdmin
        static_assert (
                    (
                    e1.msg.sender == currentPendingAdmin &&
                    currentPendingAdmin != 0 &&
                    result == 0 &&
                    newAdmin == currentPendingAdmin &&
                    newPendingAdmin == 0
                    )
                    ||
                    (
                    (
                    e1.msg.sender != currentPendingAdmin ||
                    currentPendingAdmin == 0
```

```
                    ) &&
                    result != 0 &&
                    newAdmin == currentAdmin &&
                    newPendingAdmin == currentPendingAdmin
                    )
                    );
}

// Invariant: To change admin or currentPendingAdmin, must come from current admin
rule invariantRequireAdminToChangeAdmin(address caller, address currentAdmin, address
currentPendingAdmin, address desiredAdmin, address newAdmin, address newPendingAdmin)
        description "Failed to prove that required to be admin to change admin (caller=$caller,
currentAdmin=$currentAdmin, currentPendingAdmin=$currentPendingAdmin, desiredAdmin=$desiredAdmin,
newAdmin=$newAdmin, newPendingAdmin=$newPendingAdmin)"
{
        // Free Variables
        env e0; havoc e0;
        env e1; havoc e1;
        env e2; havoc e2;
        env e3; havoc e3;

        // Strict ordering
        static_require e1.block > e0.block;
        static_require e2.block > e1.block;
        static_require e3.block > e2.block;

        static_require currentAdmin == sinvoke admin(e0);
        static_require currentPendingAdmin == sinvoke pendingAdmin(e0);

        // Start with a zero admin
        static_require currentPendingAdmin == 0;

        static_require caller == e1.msg.caller;

        // Invoke set new pending admin
        uint256 result0 = sinvoke _setPendingAdmin(e1, desiredAdmin);
        uint256 result1 = sinvoke _acceptAdmin(e2);

        static_require newAdmin == sinvoke admin(e3);
        static_require newPendingAdmin == sinvoke pendingAdmin(e3);

        static_assert (
                    e1.msg.sender == currentAdmin ||
                    (
                    newAdmin == currentAdmin &&
                    newPendingAdmin == currentPendingAdmin
```

```
                    )
                    );
}

rule _setComptroller(address desiredComptroller)
       description "Failed to set comptroller: result=$result (currAdmin=$currAdmin,
currComptroller=$currComptroller, desiredComptroller=$desiredComptroller,
nextComptroller=$nextComptroller)" {

       // Free Variables
       env e0; havoc e0;
       env e_set; havoc e_set;
    env e1; havoc e1;
       env e_accept; havoc e_accept;
    env e2; havoc e2;

       // Strict ordering
       static_require e_set.block > e0.block;
       static_require e1.block > e_set.block;
    static_require e_accept.block > e1.block;
    static_require e2.block > e_accept.block;

       address currAdmin = sinvoke admin(e0);
       address currComptroller = sinvoke comptrollerImplementation(e0);

    // Step 1: Invoke set new pending comptroller
    uint256 result_set = sinvoke _setPendingImplementation(e1,desiredComptroller);

    // Results and checks:
    address updated_pending = sinvoke pendingComptrollerImplementation(e1);
    static_assert (result_set == 0 <=> (e1.msg.sender == currAdmin && desiredComptroller ==
updated_pending)), "Mismatch in success case: result of setting pending implementation
${result_set}. Sent by ${e1.msg.sender}, current admin ${currAdmin}, wanted to set to
${desiredComptroller} with updated pending comptroller implementation is ${updated_pending}.";
    static_assert result_set == 1 <=> e1.msg.sender != currAdmin, "Mismatch in unauthorized case:
result is ${result_set} and sender is ${e1.msg.sender} when current admin is ${currAdmin}.";

       // Step 2: Invoke accept new comptroller
       uint256 result_accept = sinvoke _acceptImplementation(e_accept, desiredComptroller);

    // Results and checks:
       address nextComptroller = sinvoke comptrollerImplementation(e2);
    address finalPendingComptroller = sinvoke pendingComptrollerImplementation(e2);

       // if succeeded setting: nextComptroller == desiredComptroller <=> msg.sender ==
desiredComptroller
```

```
    static_assert result_set == 0 =>
        (result_accept == 0 <=>
               (e_accept.msg.sender == desiredComptroller &&
                      nextComptroller == desiredComptroller &&
                      nextComptroller != 0 && // Cannot set new comptroller to 0
                      finalPendingComptroller == 0)),
        "If setting pending implementation succeeded, accept will succeed (got ${result_accept})
only if desired comptroller $desiredComptroller sent the request (sent by ${e_accept.msg.sender},
and set next comptroller as ${nextComptroller}";
    static_assert result_set == 0 =>
        (result_accept == 1 <=>
               ((e_accept.msg.sender != desiredComptroller || desiredComptroller == 0) && // fails
if bad sender, or trying to effectively erase desired comptroller
               nextComptroller == currComptroller &&
               finalPendingComptroller == desiredComptroller)), "If setting pending implementation
succeeded, will fail with unauthorized (got ${result_accept}) only if different implementation tried
to accept on behalf of ${desiredComptroller} (sent by ${e_accept.msg.sender}) and did not change the
next comptroller: ${nextComptroller} from current one ${currComptroller}.";
}
```

Maximillion

repayBehalf

maximillion.cvl

```
methods {
        borrowBalance(address) returns uint256
        etherBalance(address) returns uint256

        repayBehalf(address)
}

rule repayBehalf(uint256 repayAmount, address borrower, address repayer, uint256 blockNumber)
description "Break repayBehalf" {
        env e0; havoc e0;
        env e1; havoc e1;
        env e2; havoc e2;

        static_require e0.block.number == blockNumber
                       && e1.block.number == blockNumber
                       && e2.block.number == blockNumber;

        uint256 borrowed = sinvoke borrowBalance(e0, borrower);

        static_require repayAmount == e1.msg.value
                       && repayer == e1.msg.sender;
        invoke repayBehalf(e1, borrower);
        bool repayReverted = lastReverted;

        uint256 currentBorrows = sinvoke borrowBalance(e2, borrower);
        uint256 repayerBalance = sinvoke etherBalance(e2, repayer);

        static_assert (!repayReverted =>
                       ((repayAmount >= borrowed) => (repayerBalance >= repayAmount - borrowed))),
"Mismatch in refund";
        static_assert (!repayReverted =>
                       ((repayAmount >= borrowed) => (currentBorrows == 0)) &&
                       ((repayAmount < borrowed) => (currentBorrows == borrowed - repayAmount))),
"Mismatch in borrows repaid";
}
```